Calculabilité et Complexité (11X008) - Printemps 2026

6. Relations entre classes

Enseignant: Arnaud Casteigts

Assistants: M. De Francesco, M. Marseloo
Moniteurs: E. Bussod, N. Beghdadi

6.1 Definition des classes (rappel)

Pour rappel, les deux classes de complexité génériques pour l'espace et le temps sont :

- $\mathsf{TIME}(f(n))$: Langages décidables en temps $O(f(n))$.
- $\mathsf{SPACE}(f(n))$: Langages décidables en espace $O(f(n))$.

Cas particuliers très étudiés :

- $\mathsf{LOGSPACE} = \mathsf{SPACE}(\log n)$ — Espace logarithmique
- $\mathsf{P} = \mathsf{TIME}(n^{O(1)})$ — Temps polynomial
- $\mathsf{PSPACE} = \mathsf{SPACE}(n^{O(1)})$ — Espace polynomial
- $\mathsf{EXP} = \mathsf{TIME}(2^{n^{O(1)}})$ — Temps exponentiel

Dans ce cours, nous allons considérer comme modèle les machines de Turing à deux bandes : une bande pour l'entrée et une autre pour le travail et la sortie, ainsi qu'un alphabet binaire. Les preuves données pourraient être légèrement différentes pour d'autres modèles (machines de Turing à une bande, machine RAM, etc.), cependant les résultats obtenus sont valides dans tous ces modèles.

6.2 Relations entre l'espace et le temps

Nous allons démontrer les inclusions suivantes l'une après l'autre :

$$\mathsf{LOGSPACE} \subseteq \mathsf{P} \subseteq \mathsf{PSPACE} \subseteq \mathsf{EXP}$$

6.2.1 $\mathsf{LOGSPACE} \subseteq \mathsf{P}$

"Tout langage décidable en espace logarithmique est décidable en temps polynomial."

Preuve : Soit L un langage dans LOGSPACE. Par définition, cela implique qu'il existe une constante k et une machine M telle que M décide L en utilisant au plus $k \log n$ cases mémoires sur sa bande de travail. Analysons le nombre de configurations possibles de M.

Comme toute machine de Turing, M a un nombre constant d'états dans son automate, disons k'. La tête de lecture a n positions possibles sur la bande d'entrée et $k \log n$ positions possibles sur la bande de travail. Pour le contenu des bandes, la bande d'entrée n'a qu'une valeur possible (l'entrée elle-même) et la bande de travail a 3 valeurs possibles pour chaque case : `0`, `1` et $\square$, donc $3^{k \log n}$ valeurs possibles au total. En tout, le nombre de configurations possibles est donc :

$$\#\texttt{confs} = k' \cdot n \cdot k \log n \cdot 1 \cdot 3^{k \log n}$$

Examinons les facteurs : Clairement, $k' \cdot n \cdot 1 \cdot k \log n = n^{O(1)}$. Reste à examiner $3^{k \log n} = n^{k \log 3} = n^{O(1)}$. Le nombre de configurations est donc $n^{O(1)} \cdot n^{O(1)} = n^{O(1)+O(1)} = n^{O(1)}$.

La machine M a donc un nombre polynomial de configurations. Pour conclure, observons qu'une configuration ne peut pas se répéter, car si elle se répète une fois, elle se répètera à l'infini et la machine ne terminera pas, ce qui contredit le fait que M décide L. Le temps d'exécution de M doit donc être inférieur au nombre de configurations possibles, il est donc lui-aussi polynomial et $L \in \mathsf{P}$.

En résumé, nous avons montré que $L \in$ LOGSPACE implique $L \in \mathsf{P}$. On a donc bien LOGSPACE $\subseteq \mathsf{P}$.

6.2.2 PSPACE $\subseteq$ EXP

"Tout langage décidable en espace polynomial est décidable en temps exponentiel."

Preuve : C'est exactement le même argument. Soit $L \in$ PSPACE. Il existe donc une machine M et une constante k telle que M décide L en utilisant au plus n^k cases mémoires sur sa bande de travail. Le nombre de configurations possibles d'une telle machine est :

$$\#\texttt{confs} = k' \cdot n \cdot n^k \cdot 1 \cdot 3^{n^k}$$

Clairement $k' \cdot n \cdot n^k \cdot 1 = n^{O(1)}$. Reste à analyser $3^{n^k} = 2^{n^k \cdot \log_2 3} = 2^{n^{O(1)}}$. On a donc $\#\texttt{confs} = n^{O(1)} \cdot 2^{n^{O(1)}} = 2^{n^{O(1)}}$ (potentiellement avec une constante supérieure à droite du signe égal). En utilisant le même argument que précédemment, le temps d'exécution ne peut pas excéder le nombre de configurations possibles, donc $L \in$ EXP.

En résumé, nous avons montré que $L \in \mathsf{P}$ implique $L \in$ EXP. On a donc bien $\mathsf{P} \subseteq$ EXP.

6.2.3 $\mathsf{TIME}(f(n)) \subseteq \mathsf{SPACE}(f(n))$ et donc $\mathsf{P} \subseteq \mathsf{PSPACE}$

"Tout langage décidable en temps $O(f(n))$ est décidable en espace $O(f(n))$."

Preuve : C'est évident pour n'importe quelle fonction $f(n)$. Si le temps exécution d'une machine est au plus de $f(n)$, alors cette machine ne peut utiliser au maximum que $f(n)$ cellules mémoire (chaque accès prend au moins une unité de temps).

Breaking news ! Le 25 février 2025, le chercheur Ryan Williams a publié un article [1], qui démontre que $\mathsf{TIME}(f(n)) \subseteq \mathsf{SPACE}(\sqrt{(f(n) \log f(n))})$. Par exemple, cela implique que $\mathsf{TIME}(n^2) \subseteq \mathsf{SPACE}(n \log n)$. Un progrès majeur sur une question vieille de 50 ans !

6.2.4 Bilan

C'est peut-être surprenant, mais en l'état des connaissances actuelles, on ne peut pas dire grand chose de plus. Il est théoriquement possible que $\mathsf{LOGSPACE} = \mathsf{P}$, ou que $\mathsf{P} = \mathsf{PSPACE}$, ou encore $\mathsf{PSPACE} = \mathsf{EXP}$, ce sont là des questions ouvertes. La majorité des chercheurs du domaine pensent que $\mathsf{LOGSPACE} \subsetneq \mathsf{P} \subsetneq \mathsf{PSPACE} \subsetneq \mathsf{EXP}$. C'est juste qu'on n'arrive pas à le démontrer ! Dans tous les cas, le théorème de Ryan Williams montre que dans un sens précis, l'espace est strictement "plus fort" que le temps. C'est assez logique : l'espace est *recyclable*, le temps ne l'est pas. Cela n'implique pas que $\mathsf{P} \neq \mathsf{PSPACE}$, car son théorème ne montre qu'une différence polynomiale entre les deux.

6.3 Théorèmes de hiérarchie

Ce pourrait-il, en théorie, que toutes ces classes soient égales ? Non, on sait quand même que $\mathsf{P} \neq \mathsf{EXP}$, et aussi que $\mathsf{LOGSPACE} \neq \mathsf{PSPACE}$. Plus précisément, il existe deux théorèmes généraux sur le temps d'un côté, et l'espace de l'autre.

6.3.1 Hiérarchie en temps

"Avec plus de temps, on peut résoudre plus de problèmes."

Cela peut sembler évident, encore faut-il l'établir. Concrètement, le théorème établit que pour presque toute fonction $f(n)$, [2]

$$\mathsf{TIME}(o(f(n))) \subsetneq \mathsf{TIME}(f(n) \log f(n)) \qquad \text{(Hartmanis \& Stearns, 1965)}$$

1. Ryan Williams, Simulating Time With Square-Root Space, `https://arxiv.org/abs/2502.17779`

2. Le théorème ne marche pas pour certaines valeurs trop petites de $f(n)$, par exemple si $f(n) = \log(n)$. Mais il marche pour la majorité des fonctions habituelles, comme n, n^2 ou encore 2^n.

La démonstration de ce théorème est assez technique et ressemble à la preuve de Turing pour le problème de l'arrêt. Nous ne la détaillerons pas. Pour vous donner une idée, considérons le cas particulier où l'on voudrait montrer que $\mathsf{TIME}(n) \subsetneq \mathsf{TIME}(n^2)$. Pour démontrer cela, il suffit de trouver un langage qui est dans $\mathsf{TIME}(n^2)$ mais qui n'est pas dans $\mathsf{TIME}(n)$. On peut construire facilement un tel langage en choisissant un temps intermédiaire, par exemple $n^{1.9}$ et en considérant le langage suivant :

$$D = \{(\langle M \rangle, w) \mid M(w) \text{ accepte en moins de } n^{1.9} \text{ étapes}\}$$

Ici, la taille de l'entrée est $n = |\langle M \rangle| + |w|$. La preuve se fait en deux étapes : (1) Montrer que $D \in \mathsf{TIME}(n^2)$ et (2) Montrer que $D \notin \mathsf{TIME}(n)$.

1. $D \in \mathsf{TIME}(n^2)$: C'est la partie facile, il suffit de simuler $M(w)$ pendant $n^{1.9}$ étapes de calcul. Si $M(w)$ accepte avant ce délai, on accepte. Sinon, on rejette. La simulation d'une machine de Turing n'est ralentie que d'un facteur logarithmique : cette simulation prend donc un temps $O(n^{1.9} \log n)$, ce qui est bien en $O(n^2)$, donc $D \in \mathsf{TIME}(n^2)$.
2. $D \notin \mathsf{TIME}(n)$: C'est la partie plus difficile. On suppose d'abord qu'il existe une machine M_D qui s'exécute en temps $O(n)$ et qui est capable de décider si une machine $M(w)$ accepte en moins de $n^{1.9}$ étapes. On utilise ensuite M_D pour construire une autre machine sur laquelle elle doit se tromper. La construction est similaire à celle de Turing vue au premier semestre, adaptée à une exécution en temps borné. Vous n'avez pas à connaître les détails, mais si cela vous intéresse, une preuve similaire peut être trouvée dans la section 3.1 du livre *Computational Complexity* (S. Arora et B. Barak, 2008) - `https://theory.cs.princeton.edu/complexity/book.pdf`.

6.3.2 Hiérarchie en space

Un théorème similaire existe pour l'espace. En fait, il est même plus fin :

$$\mathsf{SPACE}(o(f(n))) \subsetneq \mathsf{SPACE}(f(n))$$

L'idée de la preuve est la même, sauf que la simulation d'une machine de Turing a un surcoût en espace qui est seulement additif (et non multiplicatif, comme pour le temps), on peut donc se débarrasser du facteur logarithmique dans la partie droite du résultat.

6.4 Pourquoi P est la plus importante

La classe la plus étudiée est certainement P, l'ensemble des langages décidables en temps polynomial $n^{O(1)}$. La raison est que cette classe capture, d'un point de vue théorique, les problèmes qui peuvent être résolus efficacement (rapidement). Bien sûr, c'est sujet à discussion. On peut s'interroger, par exemple, sur l'efficacité réelle d'un algorithme qui prendrait

un temps n^{10}. Pour certaines applications où n est grand, même une complexité de n^2 peut s'avérer trop lent. Cependant, la classe P a de nombreux avantages, parmi lesquels :

- Abstraction : La classe P est la même pour tous les modèles de machine "raisonnables" (machine de Turing, machine RAM, modèles biologiques, *etc.*). La raison est que n'importe lequel de ces modèles peut simuler les autres en temps polynomial.
- Composabilité : Si un algorithme fait appel un nombre de fois polynomial à un autre algorithme qui s'exécute en temps polynomial, alors l'ensemble reste polynomial.
- Réalisme : La grande majorité des problèmes concrets qui sont dans P s'avèrent avoir de petits exposants (souvent $O(n)$ ou $O(n^2)$, parfois $O(n^3)$). On peut donc estimer que pour des tailles de problèmes raisonnables, la classe P capture bel et bien ce qui peut être résolu efficacement.

6.5 Quelques exemples

Voici une liste de problèmes. Pour chacun, nous indiquons la plus petite des 4 classes principales à laquelle on sait qu'il appartient (potentiellement non-définitif).

Entrée	Question	Classe
Liste d'éléments	Est-elle triée ?	LOGSPACE
Un graphe	Est-il connexe ?	LOGSPACE
Un graphe et un chemin de A et B	Ce chemin est-il le plus court de A vers B	P
	Ce chemin est-il le plus long de A vers B	PSPACE
Une matrice	Est-elle inversible ?	P
Un graphe	Est-il 2-colorable ?	LOGSPACE
Un graphe	Est-il 3-colorable ?	PSPACE
Un entier	Est-il premier ?	P
Un graphe	Existe-t-il un chemin qui visite exactement une fois chaque sommet et termine à son point de départ ?	PSPACE
Plateau d'échec $n \times n$ partiellement joué.	Existe-t-il une stratégie gagnante pour le joueur 1 ?	EXP
Entiers N et k	N admet-il un facteur $\leq k$?	PSPACE
Entiers N et k *(en unaire)*	N admet-il un facteur $\leq k$?	P

Nous aurons l'occasion de revenir sur certains de ces problèmes.

Subtilité : n désigne la taille de l'entrée. Lorsque l'entrée est une liste de N éléments, où chaque élément est de taille constante, alors il n'y a pas d'ambiguité, on a bien $n = O(N)$. Par contre, si l'entrée d'un problème est un nombre N, sa taille n'est pas $O(N)$. Par exemple,

la taille de 999 en décimal est 3 (et non 999). En binaire, $999 = 1111100111$, donc de taille 10. Quelle que soit la base utilisée (sauf en *unaire*), la taille est $n = O(\log N)$. Du coup, un algorithme de complexité $O(N^3)$ coûte en réalité $O(2^{3n})$. À garder en tête...